Last modified on 06/15/16 11:40:54

WebKit2 - High Level Document

WebKit2 is a new API layer for WebKit designed from the ground up to support a split process model, where the web content (JavaScript, HTML, layout, etc) lives in a separate process from the application UI. This model is very similar to what Google Chrome offers, with the major difference being that we have built the process split model directly into the framework, allowing other clients of WebKit to use it.

Why is it named WebKit2?

The somewhat pedestrian reason is that it's an incompatible API change from the original WebKit, so it will probably be installed as something like /System/Library/WebKit2.framework on Mac.

C API

WebKit2 provides a stable C-based non-blocking API that is mostly platform agnostic. In order to achieve the goal of a non-blocking API, several techniques are used to make the API usable while still providing a comprehensive set of features to the embedder. These techniques include:

- Notification style client callbacks (e.g. didFinishLoadForFrame) These inform the embedder that something has happened, but do not give them the chance to do anything about it.
- Policy style clients callbacks (e.g. decidePolicyForNavigationAction) These allow the embedder to decide on an action at their leisure, notifying the page through a listener object.
- Policy settings (e.g. WKContextSetCacheModel, WKContextSetPopupPolicy) These allow the embedder to opt into a predefined policy without any callbacks into the UIProcess. These can either be an enumerated set of specific policies, or something more fine-grained, such as a list of strings with wildcards.
- Injected code (e.g. WebBundle) Code can be loaded into the WebProcess for cases where all the other options fail. This can useful when access to the DOM is required. [Planned, but not currently implemented]

The major API classes are:

WKContextRef

- Encapsulates all pages related to specific use of WebKit. All pages in this context share the same visited link set, local storage set, and preferences.

WKPageNamespaceRef

- Encapsulates all pages that can script each other.

WKPageRef

- Basic unit of browsing. Stays the same as the main frame changes.

WKView[Ref]

- Native view that hooks into the platform's toolkit. On Windows, this wraps a HWND. On the Mac, it inherits from NSView.

Note that the requirement to be fully non-blocking requires an incompatible API break - many features of most existing WebKit APIs cannot be fulfilled in a non-blocking manner. Since we needed the API break anyway, we also took advantage of the opportunity to clean up and simplify the API.

Port-Specific APIs

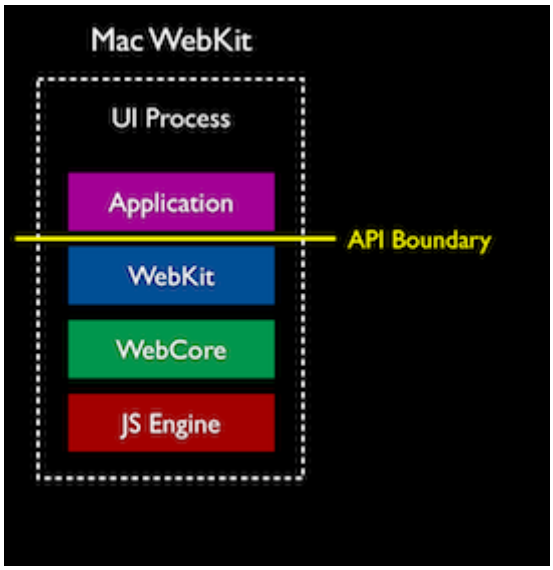
The Mac port provides a fully non-blocking Objective-C API as a wrapper on top of the C API. The GTK+ port provides also a stable and non-blocking API: http://webkitgtk.org/reference/webkit2gtk/stable/index.html

We believe a similar approach may be viable for other ports. We are also not removing or obsoleting any of the existing port-specific APIs. WebCore will remain as-is, and all current APIs will continue to work and be fully supported. Thus, WebKit development and existing ports of WebKit will not be disrupted.

Process Architecture

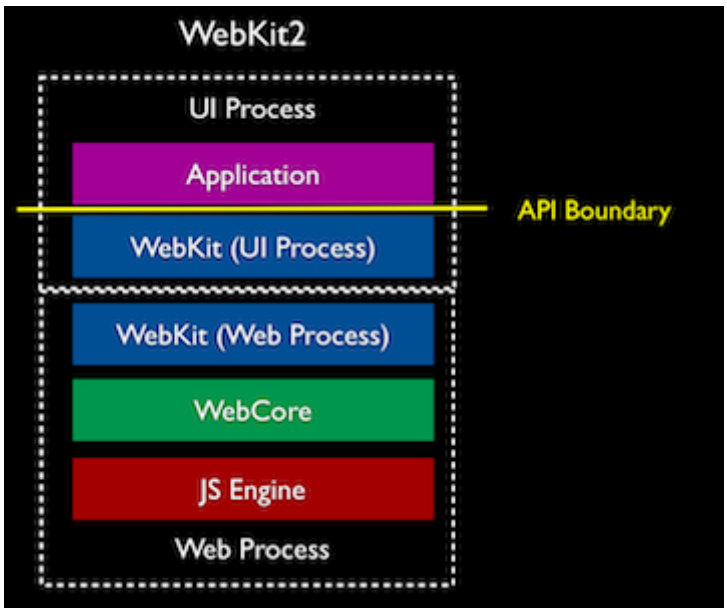
WebKit2 changes the WebKit stack to build a process management mechanism inside the WebKit API layer.

Here is what the architecture of a traditional WebKit port looks like:



Everything is in one process, and there is an API boundary between the application and the WebKit API. This is a simple model, and typically it's pretty easy for applications to reuse the WebKit API.

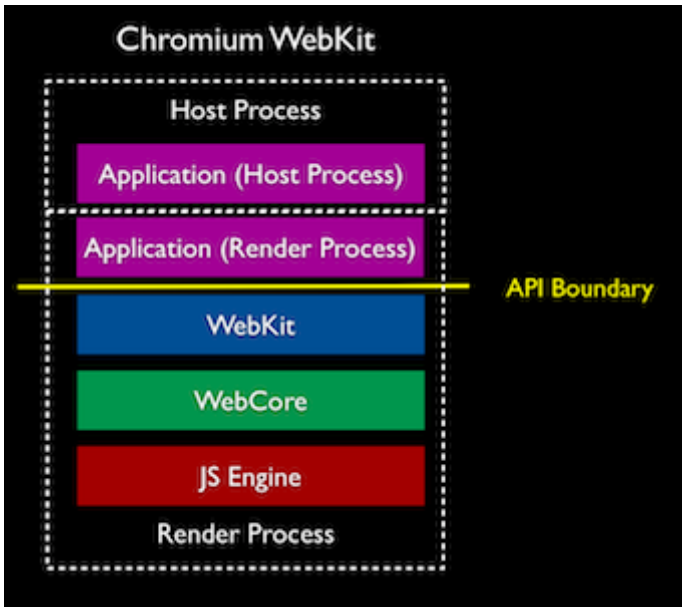
Here is what we are going for with WebKit2:



Notice that there is now a process boundary, and it sits *below* the API boundary. Part of WebKit operates in the UI process, where the application logic also lives. The rest of WebKit, along with WebCore and the JS engine, lives in the web process. The web process is isolated from the UI process. This can deliver benefits in responsiveness, robustness, security (through the potential to sandbox the web process) and better use of multicore CPUs. There is a straightforward API that takes care of all the process management details for you.

How is This Different from Chromium WebKit?

Chromium takes a different approach to multiprocess:



Notice that in this case, the the process boundary is *above* the API boundary. Chromium WebKit does not directly provide a multiprocess framework, rather, it is optimized for use as a component of a multiprocess application, which does all the proxying and process management itself. The Chrome team at Google did a great job at trailblazing multiprocess browsing with Chrome. But it's difficult to reuse their work, because the critical logic for process management, proxying between processes and sandboxing is all part of the Chrome application, rather than part of the API layer. So if another WebKit-based application or another port wanted to do multiprocess based on Chromium WebKit, it would be necessary to reinvent or cut & paste a great deal of code.

That was an understandable choice for Google - Chrome was developed as a secret project for many years, and is deeply invested in this approach. Also, there are not any other significant API clients. There is Google Chrome, and then there is the closely related Chrome Frame.

WebKit2 has a different goal - we want process management to be part of what is provided by WebKit itself, so that it is easy for any application to use. We would like chat clients, mail clients, twitter clients, and all the creative applications that people build with WebKit to be able to take advantage of this technology. We believe this is fundamentally part of what a web content engine should provide.

Internals

There are two key subsystems that support the process division :

- CoreIPC - an abstraction for general message passing, including event handling. The current implementations use mach messages on Mac OS X, and named pipes on Windows.
- DrawingArea - an abstraction for a cross-process drawing area. Multiple drawing strategies are possible, the simplest is just a shared memory bitmap.

There are two other important abstractions, which may be pushed down to WebCore or WTF over time:

- Run Loops
- Work Queues

Current Status

WebKit2 is production ready and stable. Different browsers are already using it like GNOME's Epiphany.

How to try it Out

`build-webkit` on Mac OS X or Windows now builds WebKit2 by default. WebKit2 will not work with the shipping version of Safari. Because WebKit2 is an incompatible API break, it requires a custom testbed to run it. A basic web browser application suitable for testing WebKit2 is available in Tools/MiniBrowser.

How to run layout tests

You can run layout tests in WebKit2 by passing "-2" (or "--webkit-test-runner") to run-webkit-tests, like:

```
run-webkit-tests --debug -2
```

Many tests are skipped for WebKit2 (via the LayoutTests/platform/mac-wk2/Skipped file), but that number is decreasing as DumpRenderTree API is implemented for WebKitTestRunner.

附件